WOWvx PlayerAPI

User Manual

Philips 3D Solutions

Document Information

Info	Content
Title	WOWvx PlayerAPI, User Manual
Date	2 August 2007
Security	The attached material and the information contained herein are proprietary to Philips 3D Solutions. Copying, reproduction, adaptation, modification or dissemination in whole or part is not permitted without written permission from Philips 3D Solutions.
Contact address	Philips 3D Solutions High Tech Campus 27 5656 AE Eindhoven The Netherlands, E-mail support: 3DService@philips.com E-mail general: 3DSolutions@philips.com Website: www.philips.com/3dsolutions

Table of Contents

1 Introduction 4
1.1 Document contents 4
2 WOWvx PlayerAPI distribution files 6
3 Using the WOWvx PlayerAPI 7
3.1 2D-plus-Depth and Declipse content 7
3.2 Overlays 9
3.3 Coordinates and scaling 9
3.4 WOWvx PlayerAPI concepts and data structures 10
3.5 Providing content to the WOWvx PlayerAPI 10
3.6 Typical usage of the WOWvx PlayerAPI 10
4 The WOWvxPlayerExample example application 12
4.1 Introduction 12
4.2 Setting up DirectShow 12
4.3 The Allocator/Presenter object 13
4.4 Extending the example 14
5 Troubleshooting 15

© 2007 Philips Electronics Nederland B.V.

1 Introduction

The WOWvx PlayerAPI is a library of functions that help application programmers to display 2D-plus-Depth content on a WOWvx 3D display. A large class of applications that uses this API are players of 2D-plus-Depth movies, which is where the name comes from (the prime example of one such application is the 3DSMediaPlayer that comes standard with a WOWvx display). However, other applications that want to show 3D content can also use the API.

The API is implemented in the form of a dynamic link library (.dll). It is targeted towards C/C++ applications written in Microsoft Visual Studio 6.0 or Microsoft Visual Studio 2005. It may be possible to use the API from other environments and languages, but this is not formally supported.

The API uses DirectX 9.0 to display the content. Although API itself does not support reading and decoding of movies, the example application does demonstrate how this can be done. The application needs to pass with the content to the API in the form of Direct3D textures. The API will deal with all display-related operations that are needed to show the content in 3D. This includes scaling, if the application requires this.

In the rest of the document, we assume that you are familiar with DirectX 9.0, the Microsoft Visual Studio development environment, and C/C++. For the example, we also assume that you have some familiarity with DirectShow. We also assume that you are familiar with WOWvx 3D displays, specifically that you have read the '3D Interface Specification – white paper', which can be downloaded from the 3D Solutions web site.

1.1 Document contents

Chapter 2 describes the files that come with the WOWvx PlayerAPI.

Chapter 3 introduces some basic concepts of the WOWvx PlayerAPI and provides a general description of the API.

Chapter 4 explains the example application.

Chapter 5 contains the troubleshooting guide, including the known problems.

Note that there are two more files that contain relevant information:
- inc\WOWvxPlayerAPI.h: this file contains the API reference/specification.
- inc\ReleaseNotes.h: this file contains the release notes, including some limitations of the current implementation.

© 2007 Philips Electronics Nederland B.V.

Both files are in the form of documented header files. This documentation has been converted to html files that are easier to read with Doxygen (http://www.doxygen.org, but you don't need to know about this tool in order to read the documentation). The main page is: doc\html\index.html.

© 2007 Philips Electronics Nederland B.V.

2 WOWvx PlayerAPI distribution files

The WOWvx PlayerAPI distribution consists of the following files:

inc\WowVxPlayerApi.h	The header file that contains the WOWvx PlayerAPI function headers and data types. This header file has been extensively documented and the entire API specification can be found here.
inc\ReleaseNotes.h	This header file contains the release notes of the WOWvx PlayerAPI in a comment. It has no C/C++ definitions.
doc\WOWvxPlayerAPI User Manual.pdf	This document.
doc\html*.*	This directory contains the Doxygen output for the previous two files. The file index.html is the main file.
lib\WowVxPlayerApi.dll	The Dynamic Link Library containing the object code of the API.
lib\WowVxPlayerApi.lib	The library that makes linking to the .dll file possible.
WowVxPlayerExample\Allocator.cpp	Example code for the VMR-9 Allocator/Presenter component. This file contains the code that calls the API.
WowVxPlayerExample\Allocator.h	Header file for Allocator.cpp
WowVxPlayerExample\Common.h	Header file containing common definitions. It is mostly empty.
WowVxPlayerExample\StdAfx.cpp	Visual Studio files containing headers used throughout the example project.
WowVxPlayerExample\StdAfx.h	
WowVxPlayerExample\WOWvxPlayerExample.cpp	Example code for setting up a DirectShow graph that plays a movie.
WowVxPlayerExample\WOWvxPlayerExample.dsp	Microsoft Visual Studio 6.0 project files for the example application.
WowVxPlayerExample\WOWvxPlayerExample.dsw	
WowVxPlayerExample\WOWvxPlayerExample.sln	Microsoft Visual Studio 2005 project files for the example application.
WowVxPlayerExample\WOWvxPlayerExample.vcproj	

© 2007 Philips Electronics Nederland B.V.

3 Using the WOWvx PlayerAPI

3.1 2D-plus-Depth and Declipse content

As mentioned in the introduction, the purpose of the WOWvx PlayerAPI is to display 2D-plus-Depth or Declipse content on a WOWvx compatible display. These content types are explained in the next two sections.

3.1.1 2D-plus-Depth content

2D-plus-Depth content consists of two components:

1) RGB data. This is the basic 2D information of the content. If this component is taken alone, you are left with regular 2D content.
2) Depth data (also called depth map). This component defines a depth for each pixel. The depth data has no color component. The whiter a pixel, the closer to the viewer it is.

Figure 1 shows RGB and Depth data (the RGB data on the left, the Depth data on the right). The depth map clearly shows that objects closer to the viewer are whiter.

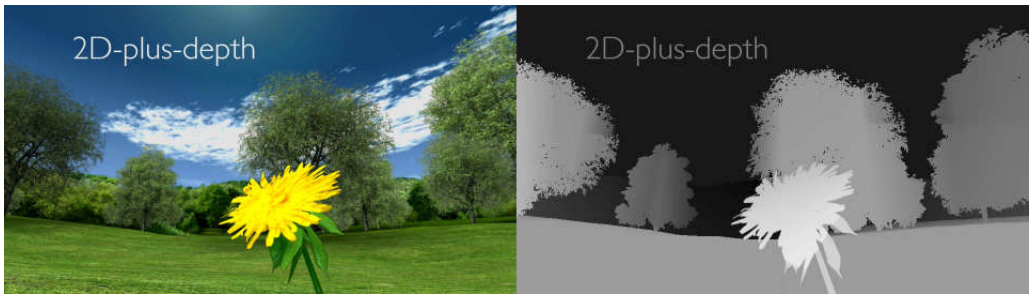

Figure 1. 2D-plus-Depth without Declipse.

3.1.2 Declipse content

Declipse content also contains the two components from 2D-plus-Depth content and it adds the following two components:

3) Declipse RGB data. Declipse data is basically an extra background layer that is used whenever the 3D display needs image data behind foreground objects, i.e. to 'look around' an object.

© 2007 Philips Electronics Nederland B.V.

4) Declipse depth data. This depth data is used both for the depth impression on the background objects and to determine where the background information can be used. The background information only makes sense if it is lower than the foreground depth. Pixels that have background depth higher than the foreground depth have no background information. Typically, areas without foreground objects are made white in the Declipse depth data. Note that at locations where no foreground object is present, the RGB data component already contains the required information, and the Declipse RGB values do no matter: they are typically left black.

Figure 2 shows content with foreground and background information. Note that there are four foreground objects (the three dragonflies and the WOWvx logo). The Declipse layers contain the background information at those locations. Also note that the area where the background is present in the Declipse layer is a bit larger than the actual size for the foreground objects. This is perfectly valid. In fact, the entire background layer could be in the Declipse layer. Erasing some of the pixels is mainly done for coding efficiency when the content is compressed. That is also why the content is blocky: the erasing is done on boundaries that fall on macroblock boundaries to reduce coding artefacts.

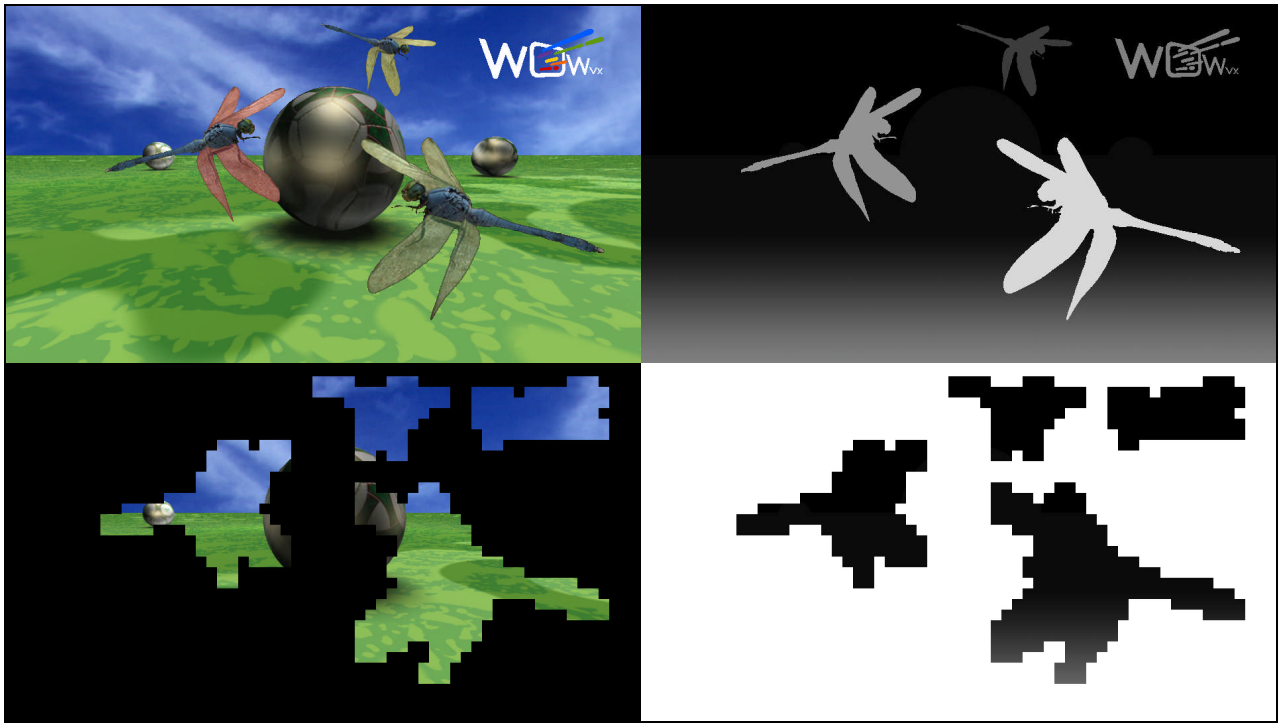

Figure 2. Declipse content with 4 components. Top-left is the RGB data, top-right is the Depth data, bottom-left is the Declipse RGB data, bottom-right is the Declipse Depth data.

More information on the 2D-plus-Depth formats and Declipse can be found in the '3D Interface Specification – white paper'.

© 2007 Philips Electronics Nederland B.V.

3.2 Overlays

The WOWvx PlayerAPI also supports up to two overlays. These overlays will be rendered on top of the main content. The purpose of these overlays is to provide for text or a logo on top of the content. An overlay cannot contain Declipse information. An overlay will always be rendered in the foreground. Please refer to the API release notes to check on the interaction between overlays and Declipse content.

The RGB data of the overlay will always be drawn on top of the content, even if the depth value of the overlay is below the depth value of the main content and the overlay should not be logically visible. The overlay RGB data typically contains an alpha channel to denote transparent parts. Note that semi-transparent alpha channels will cause strange effects and are not recommended.

The depth data of an overlay will only be used if it is higher than the depth value of the content. This means that 'high' content will 'lift up' an overlay. Note that the alpha value of the depth data will be ignored entirely. Parts where the RGB data is transparent should have a depth of 0 (if not, the content will be lifted up, which may intentionally be used for an interesting effect).

3.3 Coordinates and scaling

The WOWvx PlayerAPI fully supports scaling. It also supports placing the content and the overlays anywhere on the screen. For this, it needs a consistent coordinate system. Since the API deals with textures that can contain multiple content components (e.g. 2D-plus-Depth side-by-side, like in Figure 1) the exact meaning of width and height can become confusing. Because of this, we have chosen to align the 2D parts of the content and the screen. This mapping is only relevant for the WOWvxPlayerSetDestinationRectangle function, as all other functions deal with texture sizes.

In practice, the 2D width and height of a screen are half the width and height of the native display size[1]. So a 1920x1080 display has a 2D width and height of 960x540. Passing a rectangle of 0, 0, 960, 540 to WOWvxPlayerSetDestinationRectangle will always scale the content to full screen on this display, regardless of the texture size provided for the content. Passing a rectangle of 240, 135, 480, 270 to WOWvxPlayerSetDestinationRectangle will scale the content to a quarter screen rectangle in the centre of the screen.

The API fully supports scaling, though scaling may result in some artefacts. Checking if content will be scaled, can be done by comparing the 2D size of the content to the size of the destination rectangle. E.g. the 2D size for 2D-plus-Depth side-by-side content (like in Figure 1) of 1920x540 is 960x540. For

[1] The exception is if the screen is in 2D mode. In that case the 2D width and height are equal to the native display size. The WOWvxPlayerGetDestinationInfo function can be used to get the actual 2D width and height.

© 2007 Philips Electronics Nederland B.V.

the 1920x1080 display, this is the equal to the 2D size, so no scaling is needed to display this content full screen.

3.4 WOWvx PlayerAPI concepts and data structures

All functions of the WOWvx PlayerAPI use a handle. This handle is used to store state information for the API. A handle is linked to a Direct3D device and only textures valid for that device can be used in calls to the API. An application can create and use multiple handles, though you should check the release notes for restrictions on the usage of these handles.

All functions return an error code. A return value of 0 indicates success, in all other cases the operation failed and the function had no effect.

3.5 Providing content to the WOWvx PlayerAPI

The WOWvx PlayerAPI expects the content to be in Direct3D textures. All components can be put in the same texture (like in Figure 2), but it is also possible to supply all components in separate textures. All optional components can be left out:

- If depth information is left out, the API will assume an on-screen depth (this may not work for overlays, see the release notes).
- If Declipse information is left out, the API will assume that no background information is available.
- If Declipse depth information is left out, the API will assume on-screen depth for the background (note that this is rarely useful).

3.6 Typical usage of the WOWvx PlayerAPI

Before starting implementation with the WOWvx PlayerAPI, you should read the release notes that describe restrictions on its usage. Especially, using the API from multiple threads can be restricted by the API implementation.

An application must always start with a call to WOWvxPlayerCreate to create a handle. Typically, an application only needs one handle. Only applications that use multiple 3D displays or have special requirements would need multiple handles. When the application stops, it must call WOWvxPlayerExit for each handle.

As the next step, the application typically creates a Direct3D device for the 3D display. Creating a Direct3D device is the responsibility of the application. Usually a device is coupled to a window. It makes sense to use full-screen windows, as partial windows containing 3D content often look poor. The function WOWvxPlayerFindDisplay can be used to retrieve information about 3D displays that are connected to the system. This information is very helpful when creating Direct3D devices.

© 2007 Philips Electronics Nederland B.V.

The application must then register the Direct3D device with the API using WOWvxPlayerSet3DDevice. This is usually also a good time to do calls that apply settings that rarely change (though you have the freedom to do these calls at any time):

- WOWvxPlayerSetDestination,
- WOWvxPlayerSetDestinationType,
- WOWvxPlayerSetContentType,
- WOWvxPlayerSetDestinationRectangle.

The application will then often get into a mode where it creates (and updates) content frames. The call that is used most often at this time is WOWvxPlayerSetSource. Note that if the same textures are used throughout the lifetime of the application, the call needs to be made only once. As this function and the ones from the previous paragraph have very low overhead, calling them multiple times does no harm. For instance, the application you can call these functions for every frame you draw, if you need generic code that can deal with unexpected changes.

The WOWvxPlayerDrawScene call instructs the API to actually draw the scene. It performs the drawing operations needed to transform the input for the 3D display, but it does not do the actual Direct3D Present call that displays the results on the screen. Your application has to perform that call. The reason is that your application may have specific needs or draw to an off-screen buffer, in which case a Present call makes no sense.

If your application needs to overlay a text on the content, it can create a text texture with WOWvxPlayerCreateTextTexture. This function uses Direct3D calls to create a texture with one line of text in it. It is a convenient function that may be of use for certain applications. If your application has different needs, it has to create those textures in another way.

© 2007 Philips Electronics Nederland B.V.

4 The WOWvxPlayerExample example application

4.1 Introduction

The WOWvxPlayerExample is a very simple 3D media player. It shows both how to use the WOWvx PlayerAPI and how to use DirectShow to display a 3D movie. However, it is not a full-fledged player and it has many shortcomings that prevent it from being used as a product.

When the example starts, it asks for a 3D movie file and plays that file. When the movie has stopped, the example does not exit, but continue to show a white screen. You can stop the example by pressing Alt-F4. The 3D movie must be in the 2D-plus-Depth side-by-side format (see Figure 1). The Declipse format is not supported in this example.

The WOWvxPlayerExample resembles the VMR9Allocator example from the DirectShow SDK. Please refer to the DirectShow SDK for more details on playing back movies.

The example uses the VMR-9 DirectShow component. VMR-9 is the Video Media Renderer that uses DirectX 9 for video rendering. We have chosen this renderer, because it is relatively easy to customize. It is also the only standard DirectShow component that allows you to use DirectX 9.

If you have reasons not to use the VMR-9, you could write your own DirectShow video renderer component. It is also possible to play back movies without DirectShow, e.g. by using the Windows Media SDK to decode .wmv files and displaying them directly. However, these approaches involve a lot of work and should not be tried unless you have a lot of experience in the area of movie playback.

4.2 Setting up DirectShow

WOWvxPlayerExample.cpp is the main application file. It is a basic Win32 application, so it does not rely on libraries like MFC. If your application uses MFC or a more advanced way to deal with windows, this is certainly possible.

WinMain performs the following steps:

1) CoInitialize, MyRegisterClass, InitInstance: create a full-sized window.
2) GetFileName: create a file open dialog and use it to let the user pick a file name.
3) StartGraph: create the DirectShow graph and start movie playback (this will be detailed later).
4) Start a message loop and let the application run its course.
5) CloseGraph: stop the DirectShow graph

© 2007 Philips Electronics Nederland B.V.

StartGraph performs the following steps:
1) Create a filter graph. This is always needed for DirectShow playback.
2) Create a VMR-9 component. Note that more information on the VMR-9 can be found on MSDN (http://www.msdn.com and search for VMR-9).
3) Query the VMR-9 for an IVMRFilterConfig9 interface.
4) Set the VMR-9 in renderless mode. This will cause it to use our Allocator/Presenter component.
5) Set the VMR-9 in mixing mode. In mixing mode, the VMR-9 is easier to use by the Allocator/Presenter, so it is recommended to go to mixing mode, even if mixing multiple streams is not needed.
6) Set the Allocator/Presenter. This step will be detailed later on.
7) Add the VMR-9 to the graph.
8) Query the graph for an IMediaControl and an IMediaEventEx interface.
9) Register the application to receive events from the graph (we need this to receive the EC_COMPLETE event in WndProc).
10) Call RenderFile on the graph. DirectShow will then open the specified file and try to construct a full graph that includes the VMR-9 that we already put into the graph.
11) Call Run on the IMediaControl interface to start playback

Now DirectShow has taken over and takes care that the chosen movie is played.

Setting the Allocator/Presenter is done in 4 steps:
1) Query the VMR-9 for the IVMRSurfaceAllocatorNotify9 interface
2) Create our CAllocator (explained in 4.3)
3) Use the IVMRSurfaceAllocatorNotify9 to tell the VMR-9 that we have an Allocator/Presenter. Note that the user id parameter can be used if multiple VMR-9 objects use the same Allocator/Presenter. Such a case might be useful if the RGB data and depth data would come from different files. The MultiVMR9 example in the DirectShow SDK demonstrates how this can be done.
4) Provide the CAllocator with a reference to the VMR-9.

4.3 The Allocator/Presenter object

The CAllocator class in Allocator.cpp implements the IVMRSurfaceAllocator9 and IVMRImagePresenter9 interfaces. This allows the class to act as Allocator/Presenter for the VMR-9.

It implements the following relevant functions:
1) The constructor that creates the Direct3D device and the API handle.
2) InitializeDevice that is called by the VMR-9 to make the Allocator/Presenter ready for presenting. It must create surfaces for the VMR-9 to put the content in. At this moment important parameters such as window size are known, so this is a good place to initialize the API.
3) PresentImage that is called by the VMR-9 whenever a frame is ready and needs to be displayed. Note that the PresentHelper function does the actual drawing. Only two calls to the API are

© 2007 Philips Electronics Nederland B.V.

needed at this point, since all other parameters have already been provided. Only the WOWvxPlayerSetSource and the WOWvxPlayerDrawScene calls are needed.

The CAllocator class has some more functions, but these are not relevant for understanding the example (and they do not call the API).

4.4 Extending the example

There are a number of directions how the example can be modified to become a more serious product:

- The error handling needs to be more robust. Especially the case where the Direct3D device is lost (this happens if Ctrl-Alt-Del is pressed) needs some extra handling.
- The example stops doing anything useful when it reaches the end of file. A better approach would be to quit the application or to restart the movie.
- Currently, the example accepts only the 2D-plus-Depth side-by-side format. It should also support the Declipse format (four quadrant information like in Figure 2). Providing WOWvxPlayerTexture4Quadrants as a parameter to WOWvxPlayerSetSource will do the trick. If the type of content needs to be determined at run-time, you can use the aspect ratio as a criterion: 2D-plus-Depth content typically has an extremely long aspect ratio, 32:9 or 8:3.

© 2007 Philips Electronics Nederland B.V.

5 Troubleshooting

Problem	Possible solution
Compiled example code 'disappears' under Windows Vista	If the API is installed in its default location, under 'Program Files', Windows Vista will put the compiled files at some other location (this is Vista's virtualization feature). It is better to copy the example code somewhere else.

- 0 – 0 – 0 – 0 – 0 -

© 2007 Philips Electronics Nederland B.V.